Planning Heuristic Analysis

Problems

For implementing and exploring planning search techniques, we've implemented three different propositional logic search problems, all around the goal of moving cargo from starting airports to destination airports via planes. Air Cargo Problem 1 is the simplest of the three, involving two planes at two airports and the goal is to move cargo from each airport to the other airport. The optimal solution can be achieved in 6 steps. Air Cargo Problem 2 increases the complexity by adding on additional plane, airport, and piece of cargo, with the goal of getting the three pieces of cargo from three different airports to two destination airports using 3 planes. This problem has an optimal solution of 9 steps. Finally, Air Cargo Problem 3 increases the complexity of the problem significantly by adding an additional piece of cargo and another airport, but reduces the number of planes to two. This problem has an optimal solution of 12 steps.

Figure 1. Goals for the Air Cargo Problems.

Air Cargo Problem 1	Air Cargo Problem 2	Air Cargo Problem 3
At(C1, JFK) At(C2, SFO)	At(C1, JFK) At(C2, SFO) At(C3, SFO)	At(C1, JFK) At(C3, JFK) At(C2, SFO) At(C4, SFO)

Figure 2. Optimal solutions to the Air Cargo Problems.

Air Cargo Problem 1 (6 steps)	Air Cargo Problem 2 (9 steps)	Air Cargo Problem 3 (12 steps)
Load(C1, P1, SFO) Load(C2, P2, JFK) Fly(P1, SFO, JFK) Fly(P2, JFK, SFO) Unload(C1, P1, JFK) Unload(C2, P2, SFO)	Load(C1, P1, SFO) Load(C2, P2, JFK) Load(C3, P3, ATL) Fly(P1, SFO, JFK) Fly(P2, JFK, SFO) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Unload(C1, P1, JFK) Unload(C2, P2, SFO)	Load(C1, P1, SFO) Load(C2, P2, JFK) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Fly(P1, ATL, JFK) Unload(C4, P2, SFO) Unload(C3, P1, JFK) Unload(C2, P2, SFO) Unload(C1, P1, JFK)

Uninformed Search

First, after specifying the problems in Planning Domain Definition Language (PDDL), I experimented with the success of solving the problem with different uninformed, non-heuristic search techniques: breadth_first_search, breadth_first_tree_search, depth_first_graph_search, depth_limited_search, and uniform_cost_search. I quickly eliminated depth_limited_search and breadth_first_tree_search because both allow for revisiting of states which resulted in planning that wasn't able to find a solution before I aborted the searches. For the remaining three algorithms, as expected both breadth_first_search and uniform_cost_search were able to find the optimal solution in roughly the same amount of time while evaluating similar search spaces It should be noted that uniform_cost_search performs better with respect to time for Air Cargo Problems 2 & 3 because of a difference of queues used in the respective implementation. Normally, we'd expect breadth_first_search to perform faster.

Depth_first_graph_search, while very fast compared to the other two, does come up with correct solutions, but in every case these solutions are far from optimal. It also has a much smaller search space. When considering the real-world implications of these problems, these benefits of speed and smaller search space definitely do not outweigh the costs of labor and morale of repeatedly loading and unloading the same piece of cargo many times before actually flying it to the cargo's destination.

Figure 3: Results from uninformed searches

Problem	Search Function	Expansions	Goal Tests	New Nodes	Plan Length	Time
1	breadth_first_search	43.0	56.0	180.0	6.0	0.052134
1	depth_first_graph_search	21.0	22.0	84.0	20.0	0.018530
1	uniform_cost_search	55.0	57.0	224.0	6.0	0.081317
2	breadth_first_search	3343.0	4609.0	30509.0	9.0	23.547821
2	depth_first_graph_search	624.0	625.0	5602.0	619.0	5.016421
2	uniform_cost_search	4852.0	4854.0	44030.0	9.0	21.607970
3	breadth_first_search	14663.0	18098.0	129631.0	12.0	164.912724
3	depth_first_graph_search	408.0	409.0	3364.0	392.0	2.808046
3	uniform_cost_search	18223.0	18225.0	159618.0	12.0	87.200490

Figure 4: Comparison of times for uninformed search strategies

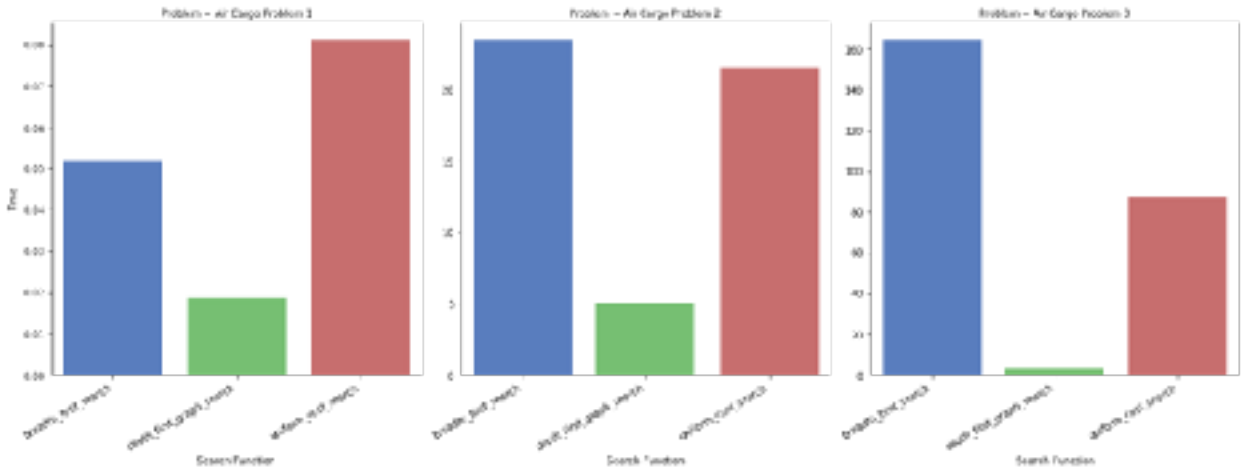


Figure 5: Comparison of plan lengths for uninformed search strategies

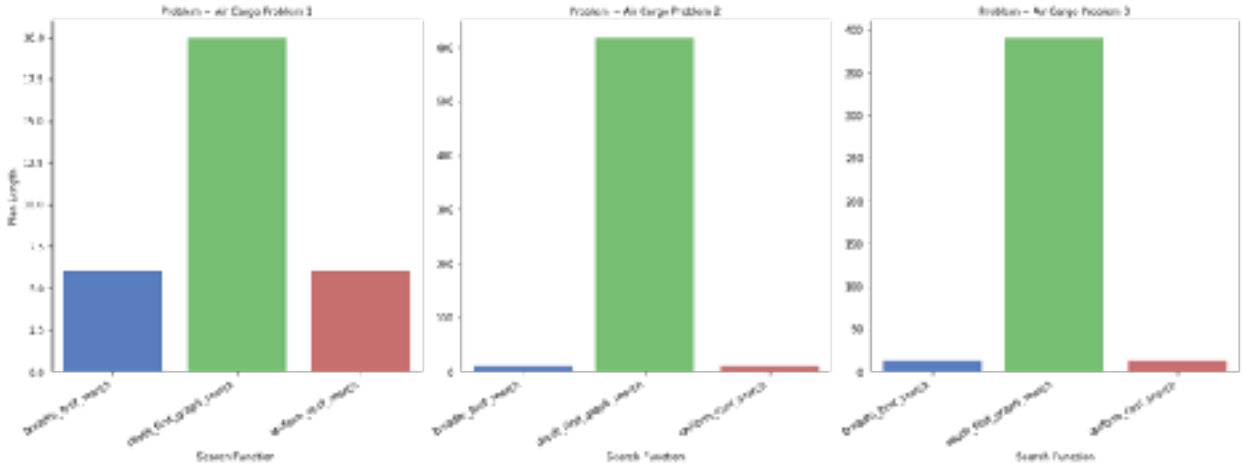
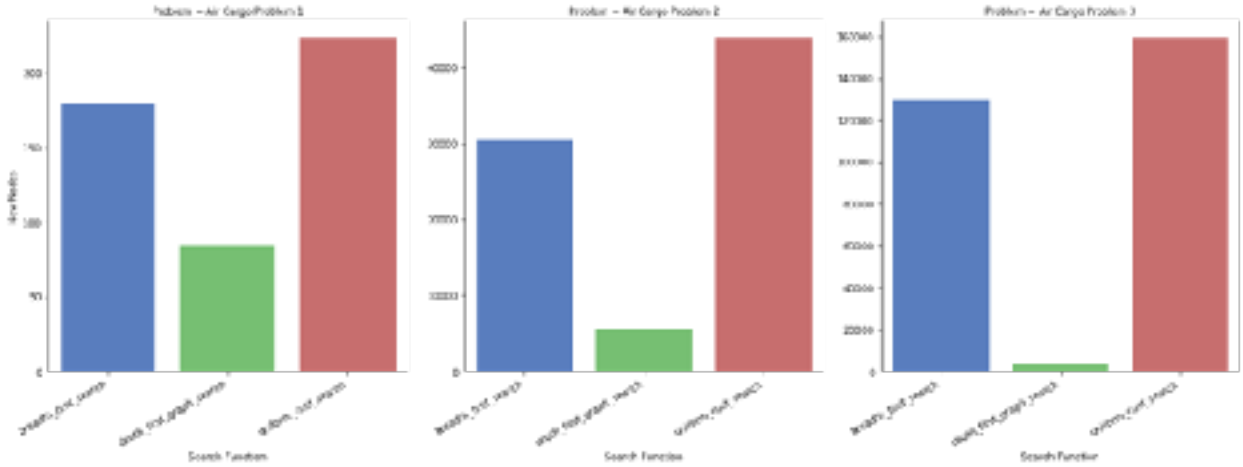


Figure 6: Comparison of new nodes for uninformed search strategies



Domain-independent heuristics

While we were able to achieve decent success with uninformed search strategies, the search space was quite large, ~150,000 nodes, for the two algorithms that were able to find optimal solutions. If we were to expand the problem set to include a problem more typical of a real-world scenario, moving hundreds of thousands of packages across hundreds of airports, utilizing thousands of airplanes, the search space would quickly make these approaches untenable. Time might also become a factor, but search space definitely would. With this in mind, I explored adding some domain-independent heuristics—heuristics that only utilize information about the abstract problem, e.g. goal clauses—primarily driven by the A* search algorithm to explore what benefits and/or tradeoffs this approach could yield.

In terms of being able to come up with an optimal plan, all of the A* searches were able to find an optimal solution. The greedy_best_first_graph_search, while very quick, was not. Although, it should be

noted it wasn't as far off of the optimal solution compared to the uninformed searches that didn't achieve an optimal solution. The A* search with a no-op heuristic was able to find the optimal solution for all problems. The problem, though, was that it searched the largest search space, creating 159,618 new nodes for problem 3, just as many as the uniform cost search, so we haven't made any progress on that front. Adding more complex heuristics, though, ignore_preconditions, which counts the minimum number of minimum number of actions required to reach the goal if you ignore the preconditions for the actions, still reaches the optimal solution and constrains the search space by 23%, reducing the number of new nodes to 123,148. This comes at the cost, though, of more than doubling the time required to find the solution. Finally, implementing a planning graph and using a levelsum heuristic that returns the sum of the level cost for each of the goal clauses reduces the search space significantly to only creating 3,002 new nodes, 2% of the nodes created using the no-op heuristic. It still comes at an increased time cost, though, requiring 356 seconds to find the optimal solution. This is expected since constructing a planning graph is polynomial in the size of the problem, $O(n(a + l)^2)$, where n is the number of levels in the graph, a is the number of actions and l is the number of literals, and has the same time complexity. This is much smaller than the exponential cost of something like the uniform-cost search, which has $O(b^{1+[C^*/\epsilon]})$ complexity for time and space[1].

Figured 7: Results from domain-independent heuristic searches

Problem	Search Function	Expansions	Goal Tests	New Nodes	Plan Length	Time
1	greedy_best_first_graph_search h_1	7.0	9.0	28.0	6.0	0.008310
1	astar_search h_1	55.0	57.0	224.0	6.0	0.059239
1	astar_search h_ignore_preconditions	42.0	44.0	176.0	6.0	0.116354
1	astar_search h_pg_levelsum	11.0	13.0	50.0	6.0	0.803304
2	greedy_best_first_graph_search h_1	990.0	992.0	8910.0	17.0	3.322261
2	astar_search h_1	4852.0	4854.0	44030.0	9.0	16.736934
2	astar_search h_ignore_preconditions	3486.0	3488.0	31795.0	9.0	32.728529
2	astar_search h_pg_levelsum	86.0	88.0	841.0	9.0	65.131005
3	greedy_best_first_graph_search h_1	5578.0	5580.0	49150.0	22.0	22.839988
3	astar_search h_1	18223.0	18225.0	159618.0	12.0	81.466148
3	astar_search h_ignore_preconditions	13903.0	13905.0	123148.0	12.0	188.678938
3	astar_search h_pg_levelsum	325.0	327.0	3002.0	12.0	356.261504

Figure 8: Comparison of time for uninformed search strategies

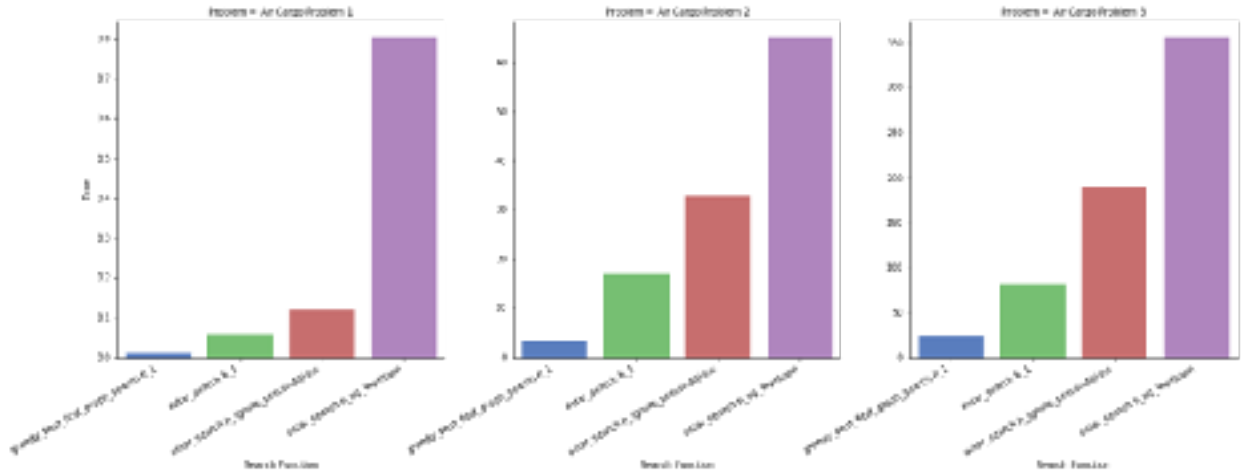


Figure 9: Comparison of plan lengths for uninformed search strategies

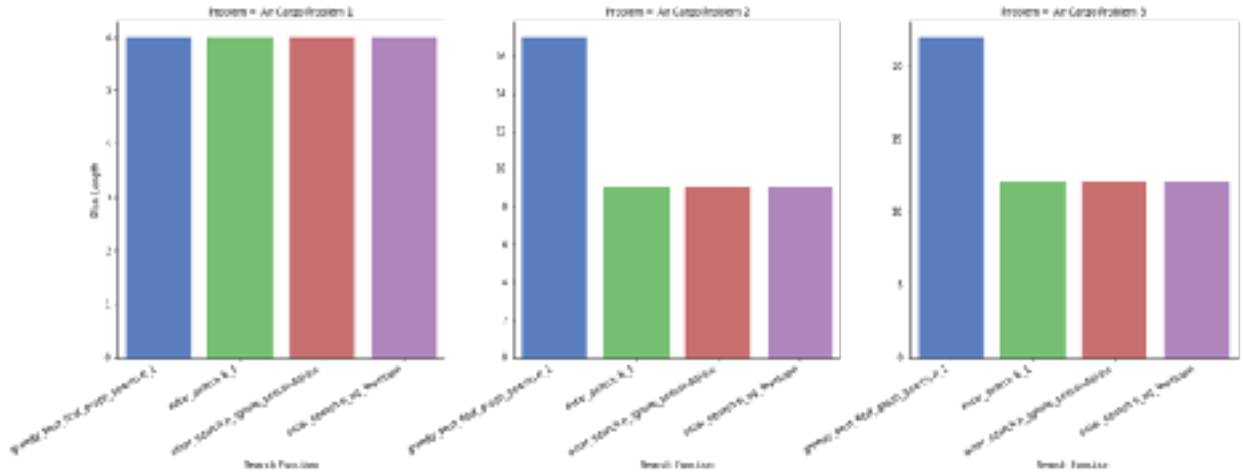
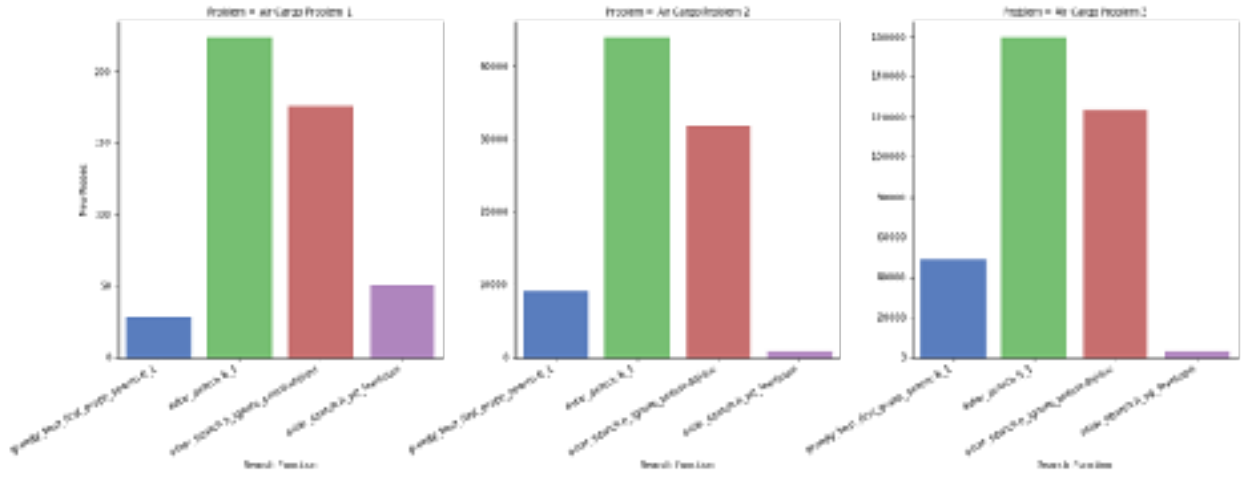


Figure 10: Comparison of new nodes for uninformed search strategies



Conclusion

Planning is a fascinating area of artificial intelligence! Using propositional logic and a formal problem definition language, you can frame many real-world problems into a set of states, actions, and goals and then use common search strategies to find solutions. There isn't a clear winner in terms of what technique and algorithm you should use, though. Depending on the type of problem, you may want to optimize for time at the expense of a larger search space. Or, you might be willing to trade optimality for a small search space and quick performance. It really depends on the needs of a given problem. For the problem of air cargo, I would reach for reducing the search space at the expense of more time, electing to use the A* search with the levelsum heuristic, since in the real-world, the actual search space requirements of all of the other algorithms would prove to be untenable.

References

1. Russell, S. J., Norvig, P., & Canny, J. (2003). Artificial intelligence: A modern approach.